Atelier Data Science

Deep learning practice 3

Optimization Techniques in DL

Irina Proskurina

Irina.Proskurina@univ-lyon2.fr

Laboratoire ERIC – Université Lyon 2

Stochastic Gradient Descent

Stochastic Gradient Descent

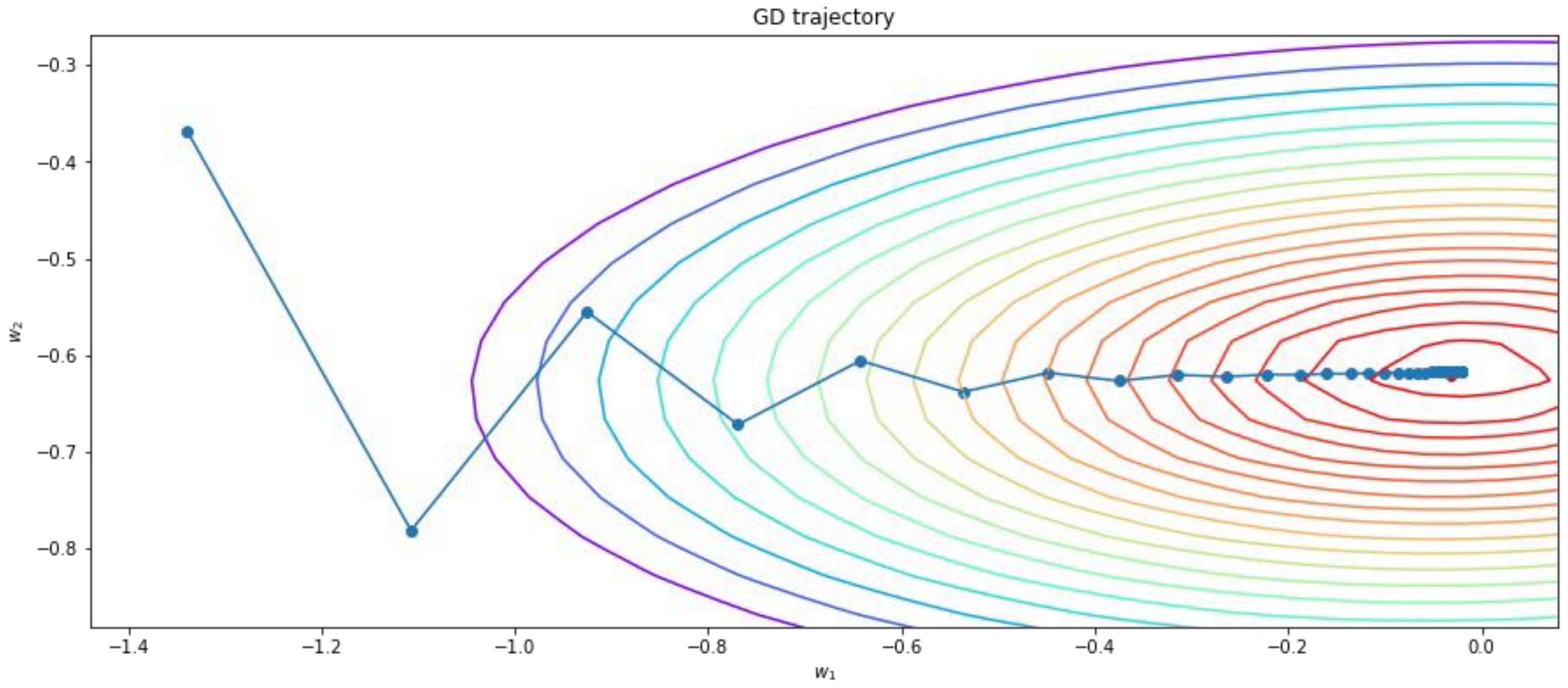
1. Initial approximation w^0
2. Repeat, each time choosing a random object i_t:

$$w^t = w^{t-1} - \eta \nabla L\left(y_{i_t}, a(x_{i_t})\right)$$

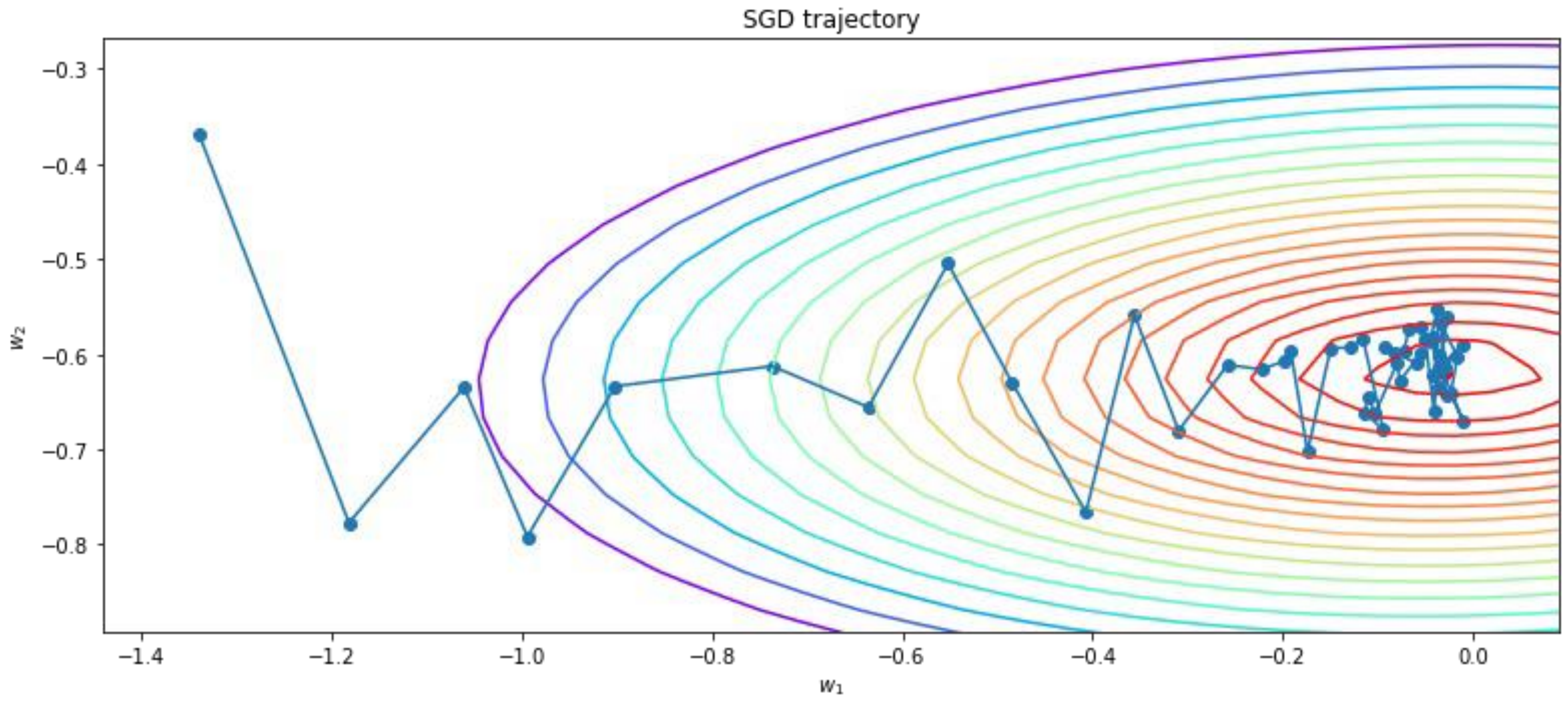
3. Stop when the error on the test set stops decreasing

```
torch.optim.SGD(params, lr=<required parameter>, momentum=0, dampe
ning=0, weight_decay=0, nesterov=False, *, maximize=False, foreach
=None, differentiable=False
```

Gradient Descent



Stochastic Gradient Descent



Stochastic Gradient Descent

1. Initial approximation w^0
2. Repeat, each time choosing a random object i_t:

$$w^t = w^{t-1} - \eta \nabla L\left(y_{i_t}, a\left(x_{i_t}\right)\right)$$

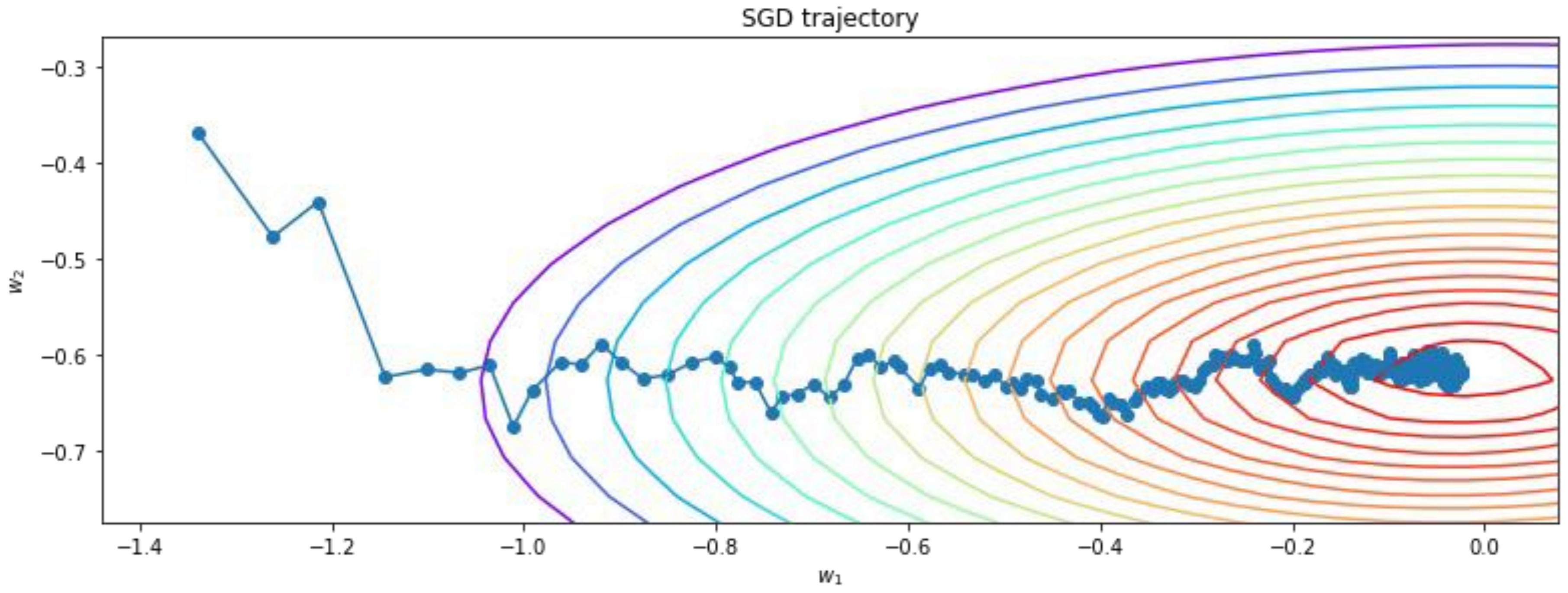
3. Stop when the error on the test set stops decreasing

Stochastic Gradient Descent

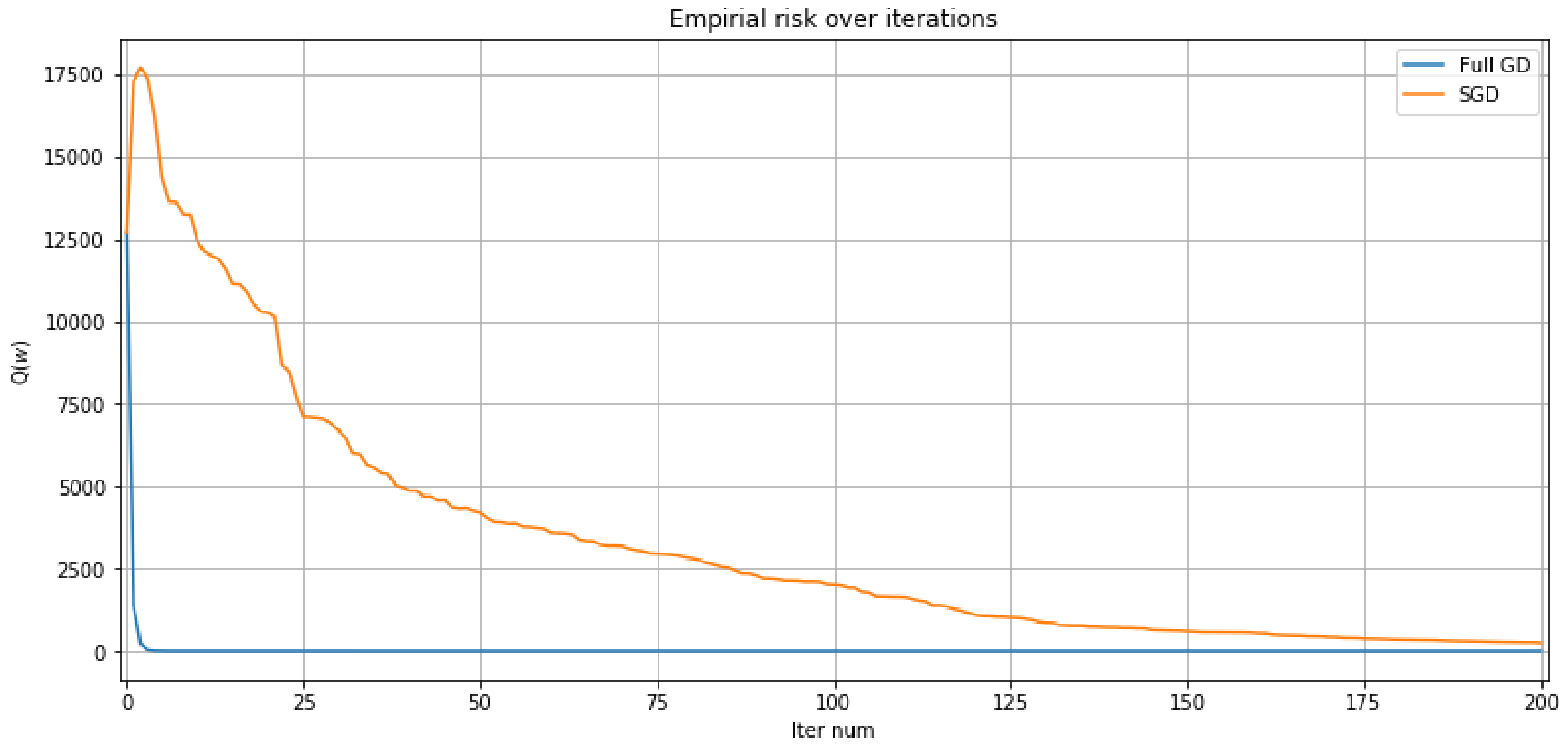
- In stochastic gradient descent, an estimate of the gradient for a single iteration is considered to be **unbiased**
- **Nonzero Gradient at Optimum**: Even when the optimization process reaches the optimum point, the gradient (estimate of the derivative of the loss with respect to the parameters) for a single data point is unlikely to be exactly zero
- **Importance of Small Learning Rates**: Because the estimate for a single object is unlikely to be precisely zero, it suggests that the optimization process is continuously refining the model parameters even at the optimum. A smaller step size allows for more precise adjustments and avoids overshooting the optimum
- **Convergence to the global minimum** is guaranteed only for convex functions

Stochastic Gradient Descent

$$\eta_t = \frac{0.1}{t^{0.3}}$$



Stochastic Gradient Descent



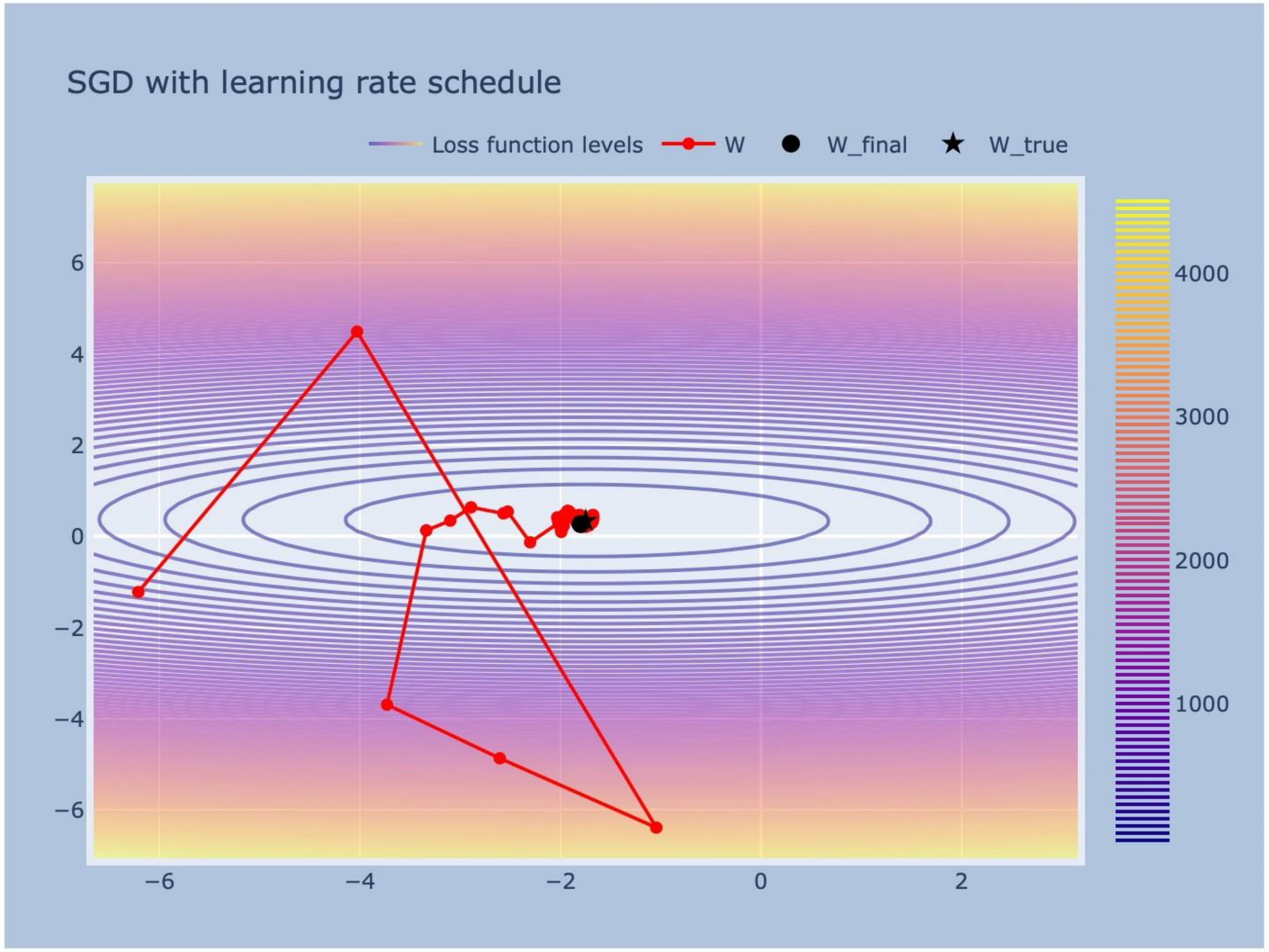
Mini-batch GD

1. Initial approximation w^0
2. Repeat, each time choosing a set of *n* random objects :

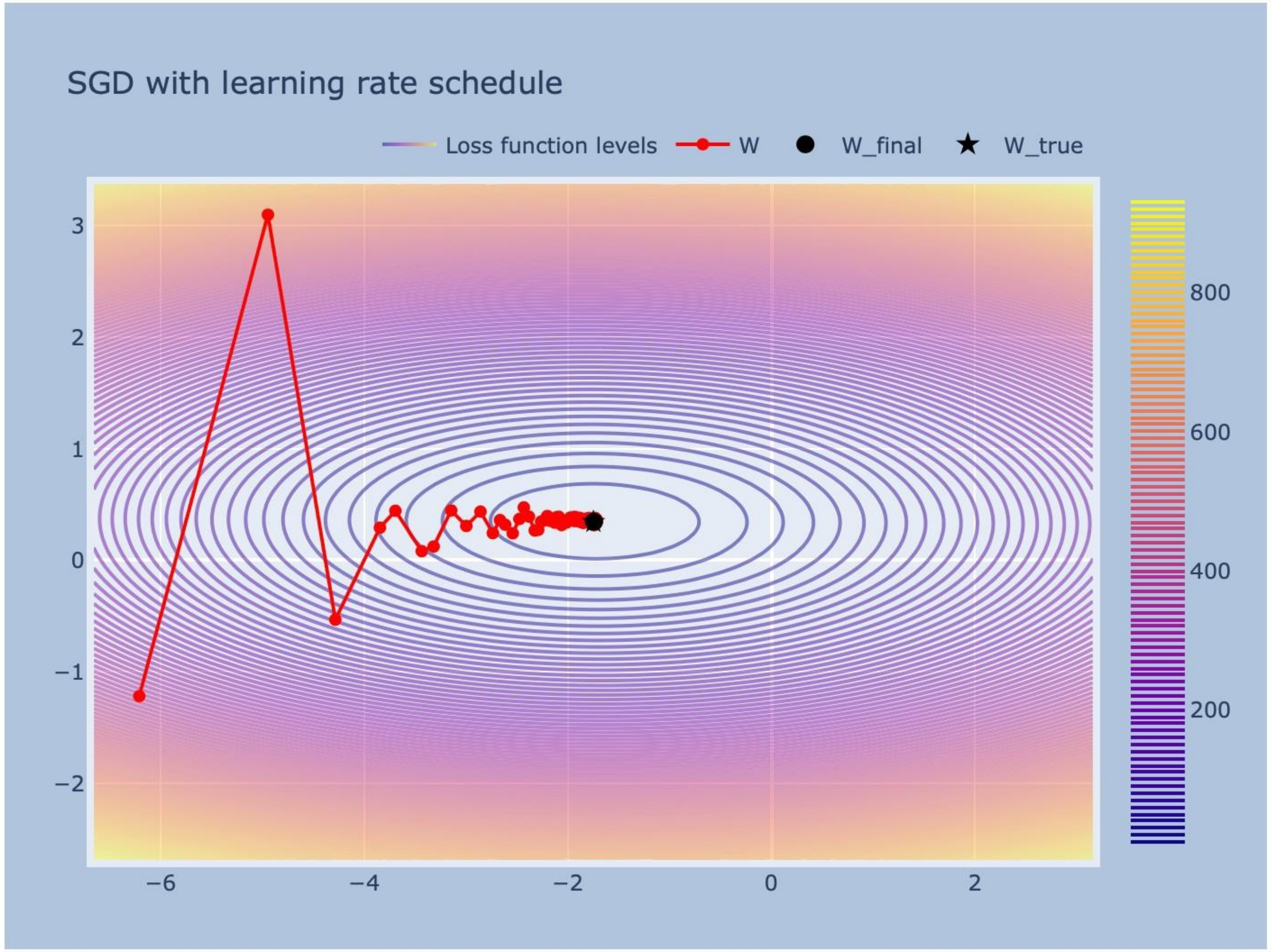
$$w^t = w^{t-1} - \eta_t \frac{1}{n} \sum_{j=1}^{n} \nabla L\left(y_{t,j}, a\left(x_{t,j}\right)\right)$$

3. Stop when the error on the test set stops decreasing

Mini-batch of size 1



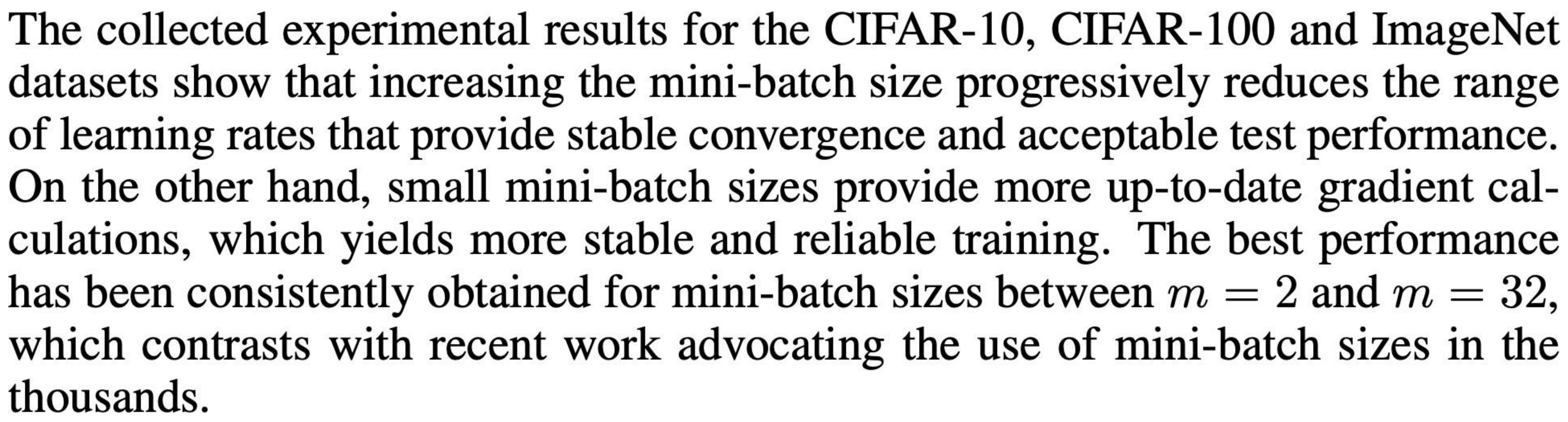
Mini-batch of size 10



Batch size

- Batch Size - typically in the order of tens or hundreds.
- It makes sense to choose a power of two.
- Computationally almost as efficient as a gradient step for a single object

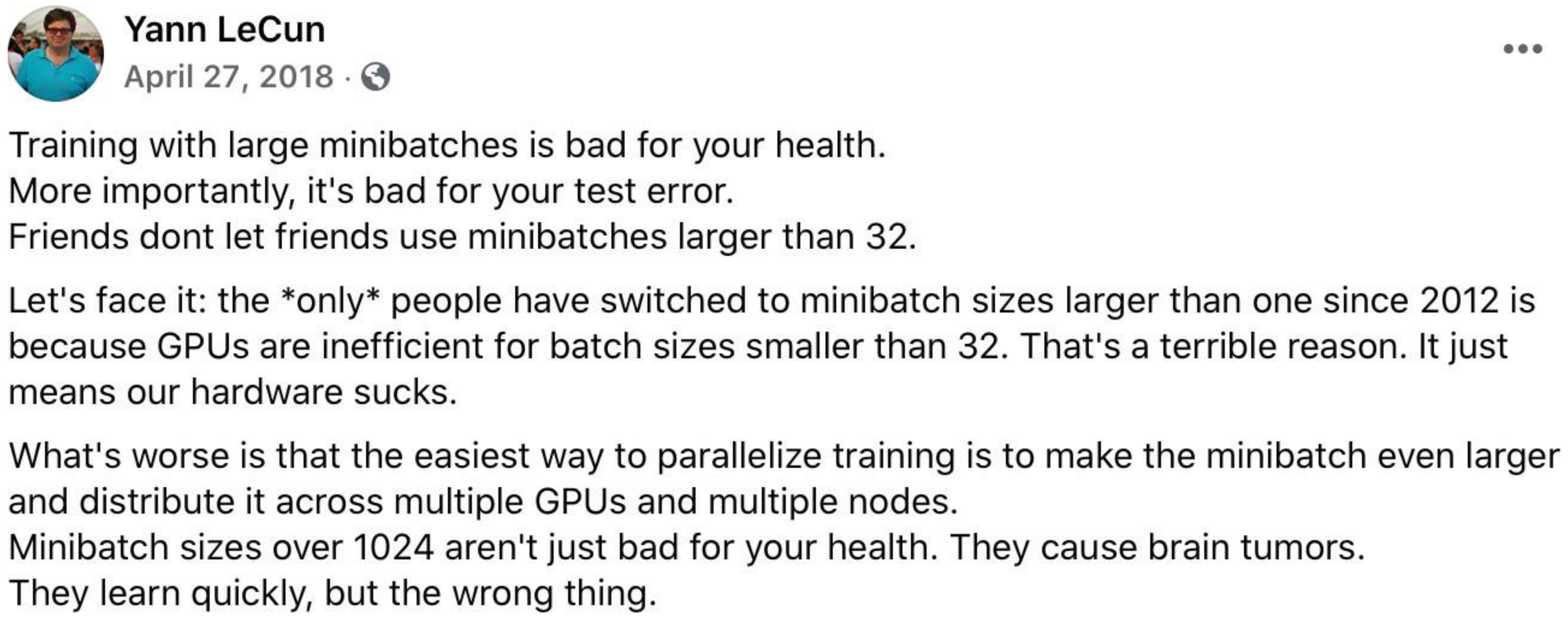
Batch size



The collected experimental results for the CIFAR-10, CIFAR-100 and ImageNet datasets show that increasing the mini-batch size progressively reduces the range of learning rates that provide stable convergence and acceptable test performance. On the other hand, small mini-batch sizes provide more up-to-date gradient calculations, which yields more stable and reliable training. The best performance has been consistently obtained for mini-batch sizes between $m = 2$ and $m = 32$, which contrasts with recent work advocating the use of mini-batch sizes in the thousands.

https://arxiv.org/abs/1804.07612

Batch size



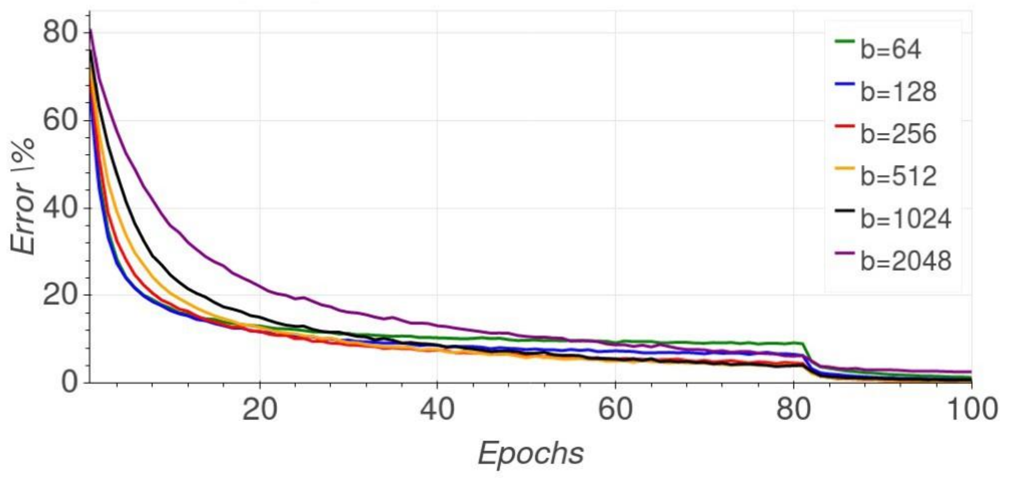
Yann LeCun
April 27, 2018 ·

Training with large minibatches is bad for your health.
More importantly, it's bad for your test error.
Friends dont let friends use minibatches larger than 32.

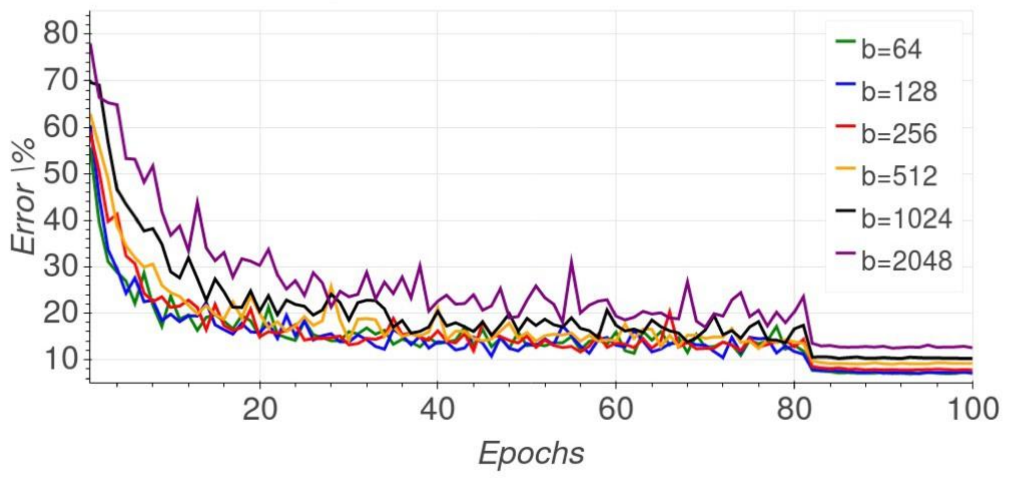
Let's face it: the *only* people have switched to minibatch sizes larger than one since 2012 is because GPUs are inefficient for batch sizes smaller than 32. That's a terrible reason. It just means our hardware sucks.

What's worse is that the easiest way to parallelize training is to make the minibatch even larger and distribute it across multiple GPUs and multiple nodes.
Minibatch sizes over 1024 aren't just bad for your health. They cause brain tumors.
They learn quickly, but the wrong thing.

Batch size



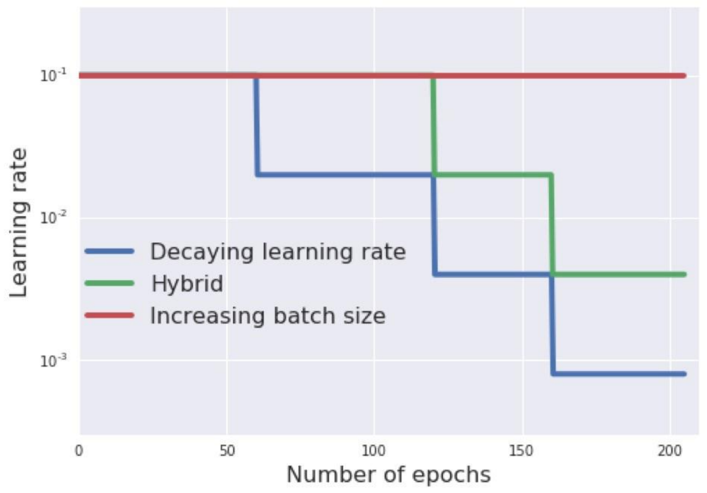
(a) Training error

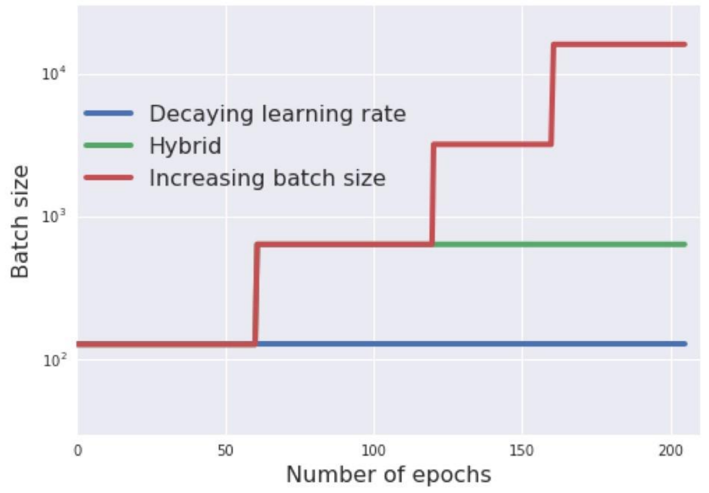


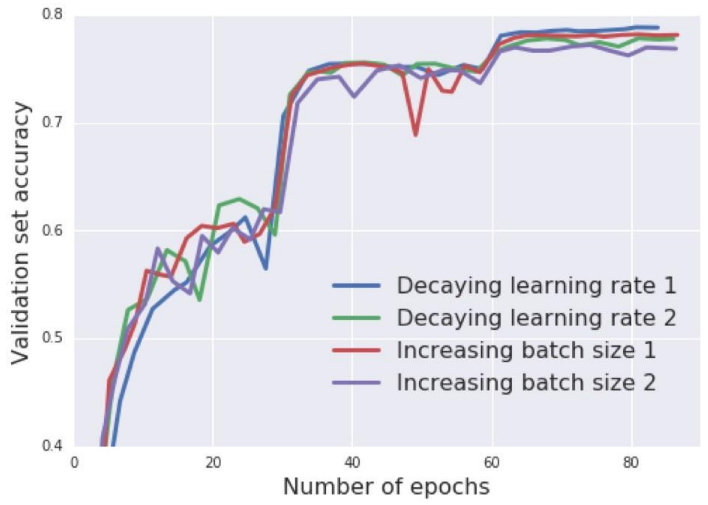
(b) Validation error

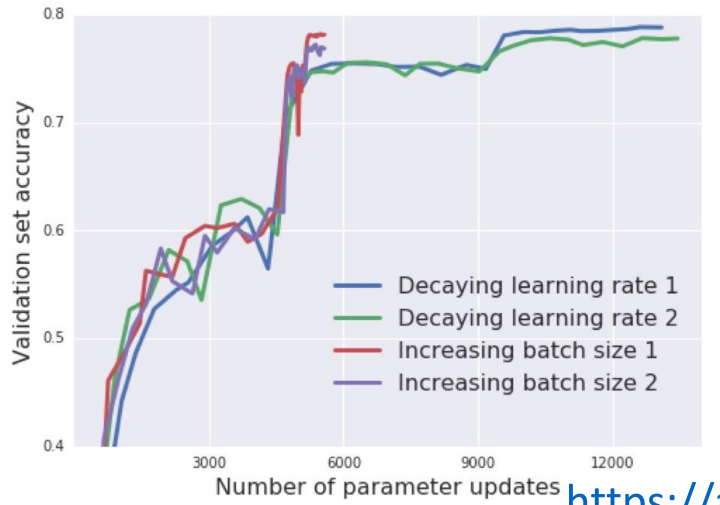
https://arxiv.org/pdf/1705.08741.pdf

Batch size









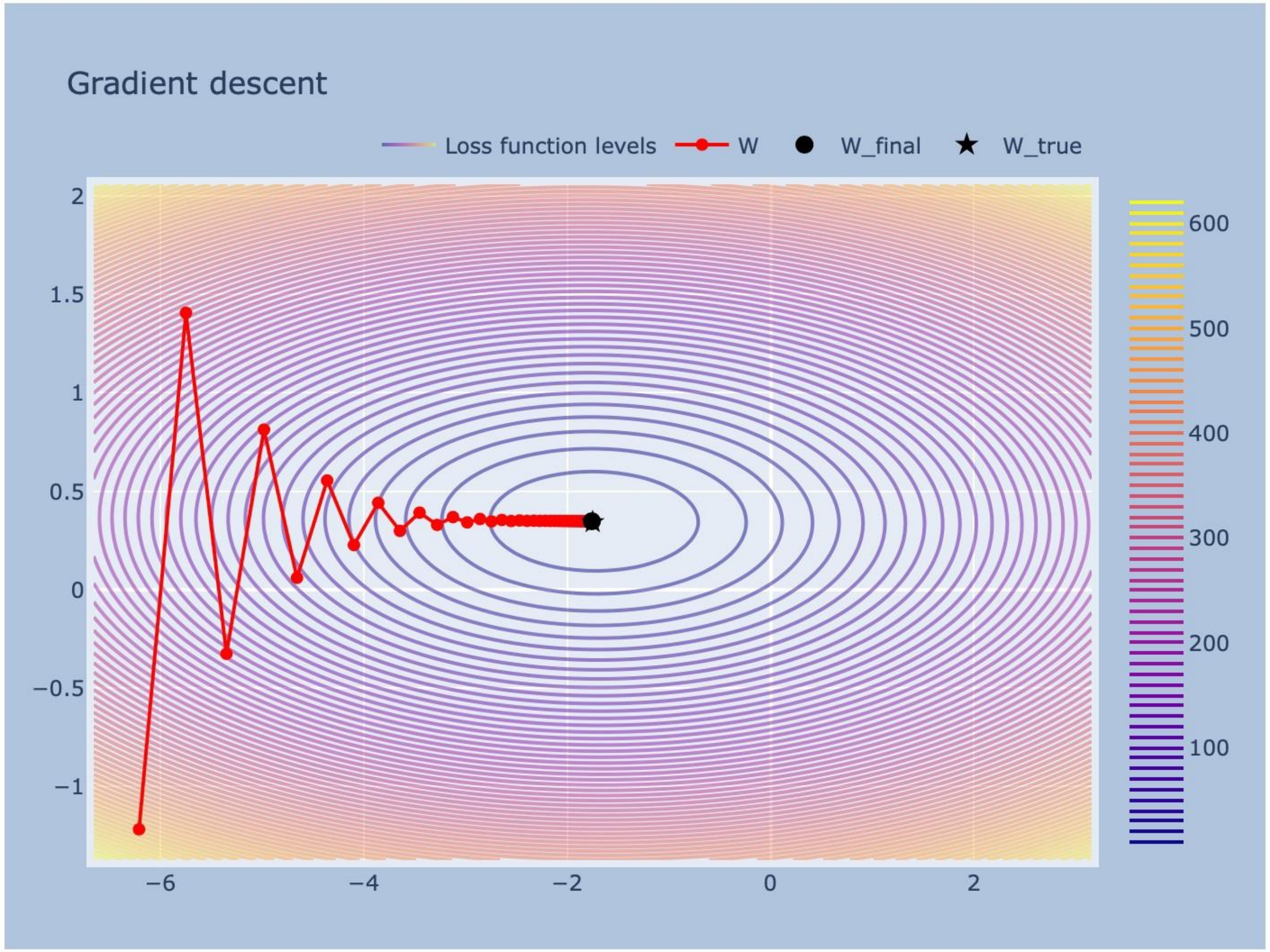
https://arxiv.org/pdf/1711.00489.pdf

Batch size

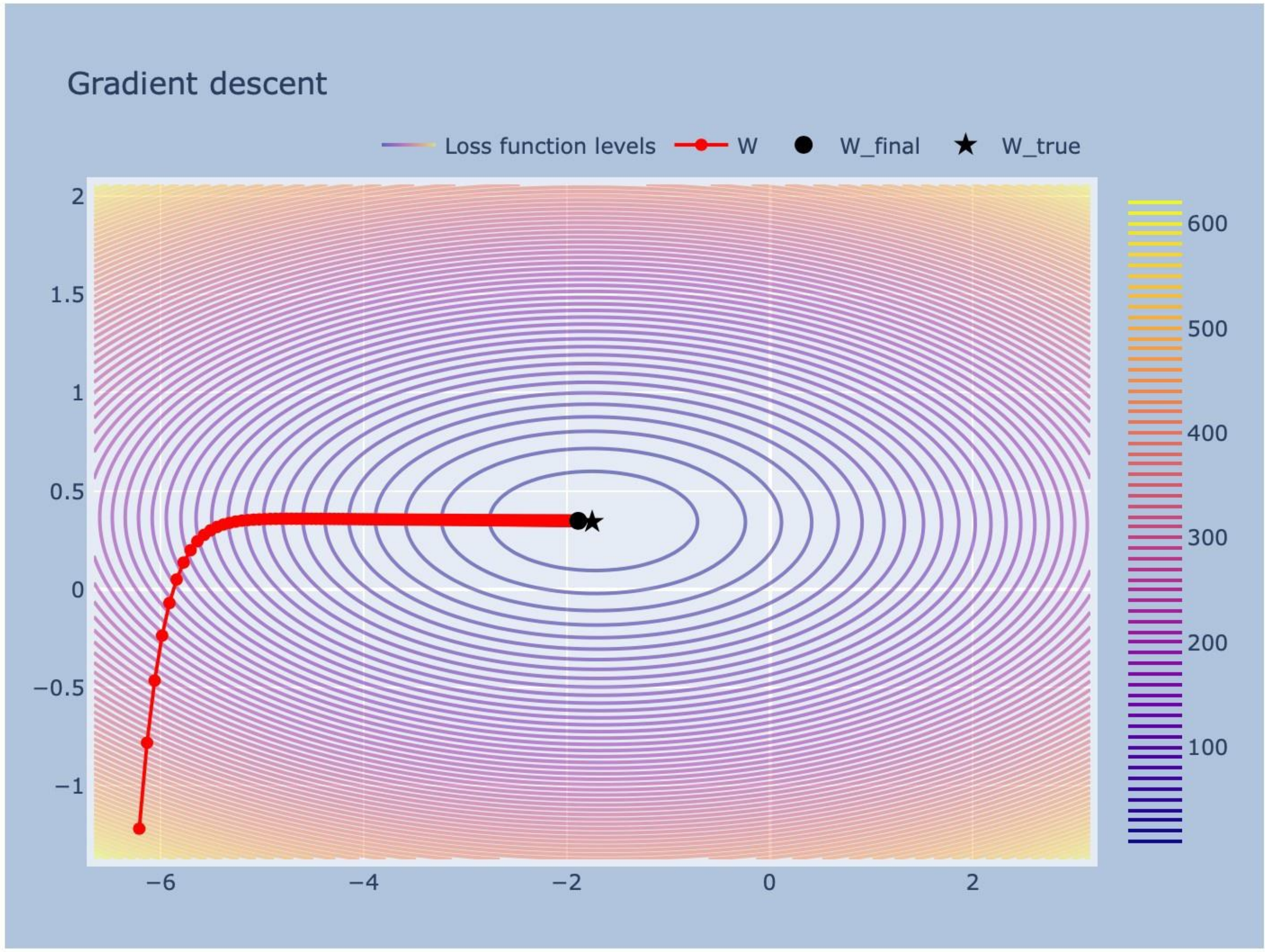
- There can be benefits from a large batch size
- To achieve this, it is necessary to choose the step size wisely and/or gradually increase the batch size

Modifications of Gradient Descent

Problems of Gradient Descent



Problems of Gradient Descent



Problems of Gradient Descent

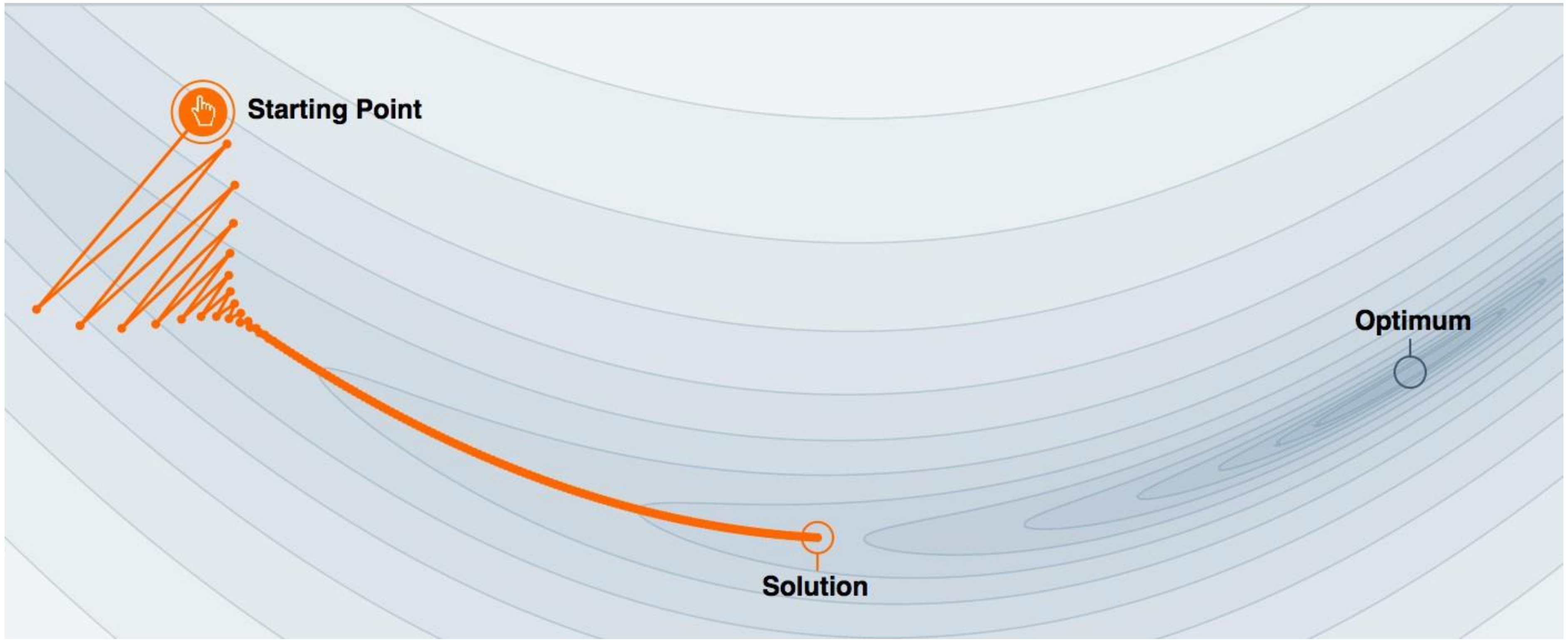
- Gradient descent is slow to converge in case of “zig-zag”

Momentum (smooths out the zig-zag path)

$$h_t = \alpha h_{t-1} + \eta_t \nabla Q(w^{t-1})$$
$$w^t = w^{t-1} - h_t$$

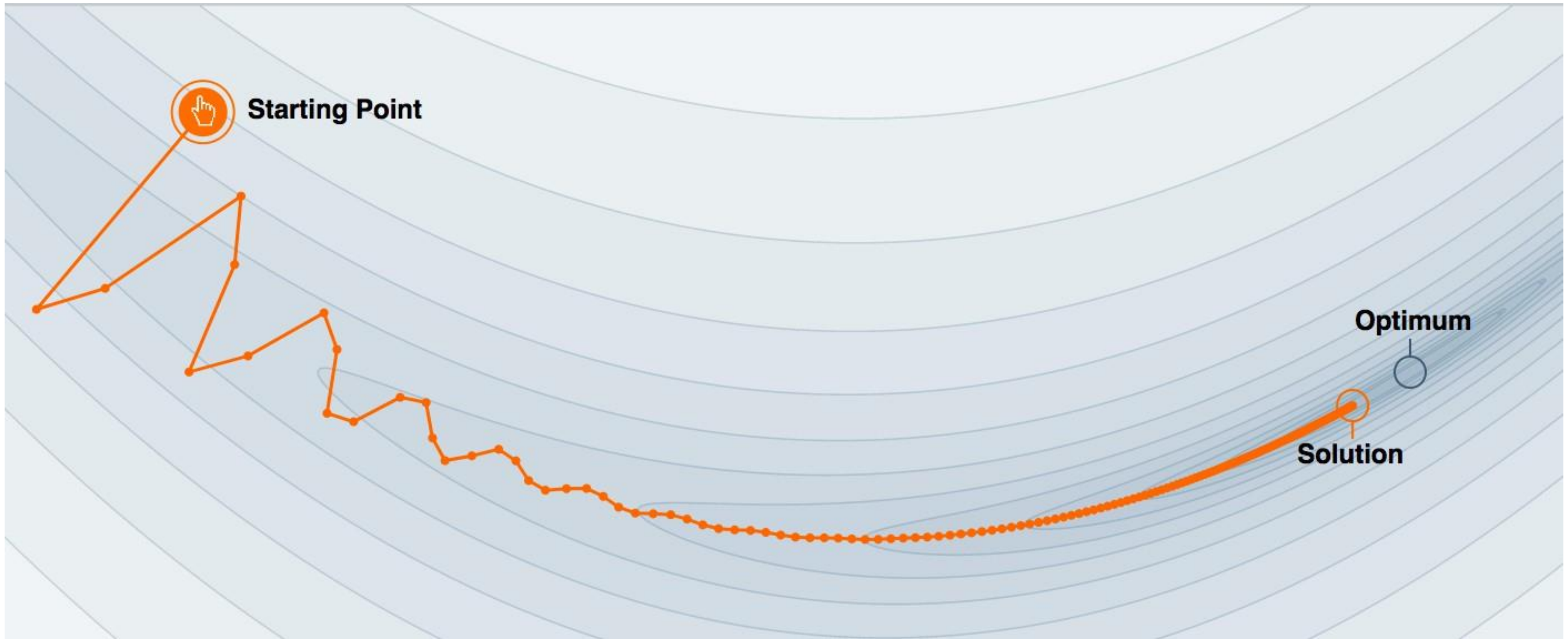
- h_t — is the velocity at time t-1
- α — momentum term

Without momentum



https://distill.pub/2017/momentum/

Momentum



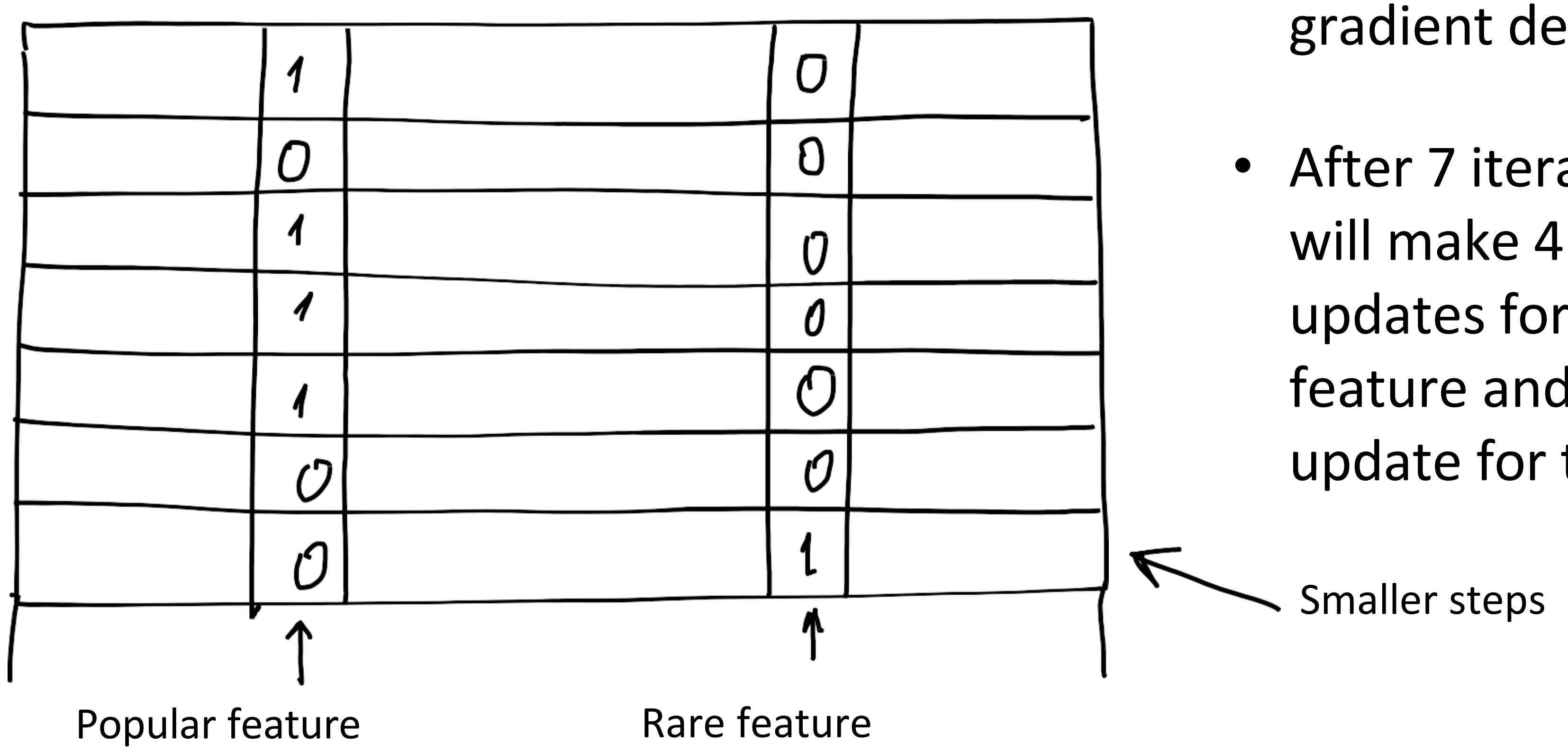
https://distill.pub/2017/momentum/

Problems of Gradient Descent

- Gradient descent is slow to converge in case of “zig-zag” -> Momentum
- Sparse Features problems

Sparse Feature problems

- Example: one-hot encoded features



- We perform stochastic gradient descent
- After 7 iterations, we will make 4 weight updates for the popular feature and 1 weight update for the rare one

$$\eta_t = \frac{0.1}{t^{0.3}}$$

AdaGrad

$$G_j^t = G_j^{t-1} + \left(\nabla Q(w^{t-1})\right)_j^2$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{G_j^t + \epsilon}} \left(\nabla Q(w^{t-1})\right)_j$$

- For each parameter, there is its own learning rate can be fixed
- The step size may decrease too rapidly and lead to early stopping

RMSProp

$$G_j^t = \alpha G_j^{t-1} + (1-\alpha)\left(\nabla Q(w^{t-1})\right)_j^2$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{G_j^t + \epsilon}}\, g_{tj}$$

- $\alpha \sim 0.9$
- The speed update depends only on recent steps

Adam

$$m_j^t = \frac{\beta_1 m_j^{t-1} + (1-\beta_1)\big(\nabla Q(w^{t-1})\big)_j}{1-\beta_1^t}$$

$$v_j^t = \frac{\beta_2 v_j^{t-1} + (1-\beta_2)\big(\nabla Q(w^{t-1})\big)_j^2}{1-\beta_2^t}$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{v_j^t + \epsilon}}\, m_j^t$$

- Recommended values: $\beta_1 = 0.9$, $\beta_2 = 0.999$
  $\epsilon = 10^{-8}$

Adam

- Momentum + RMSProp

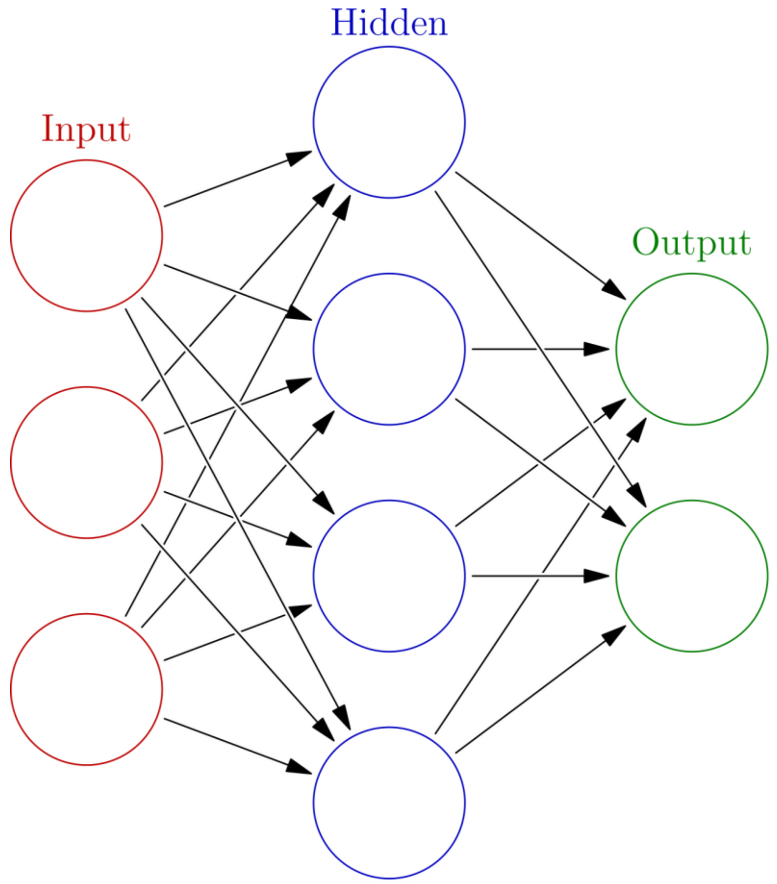
Dropout

Mitigating overfitting

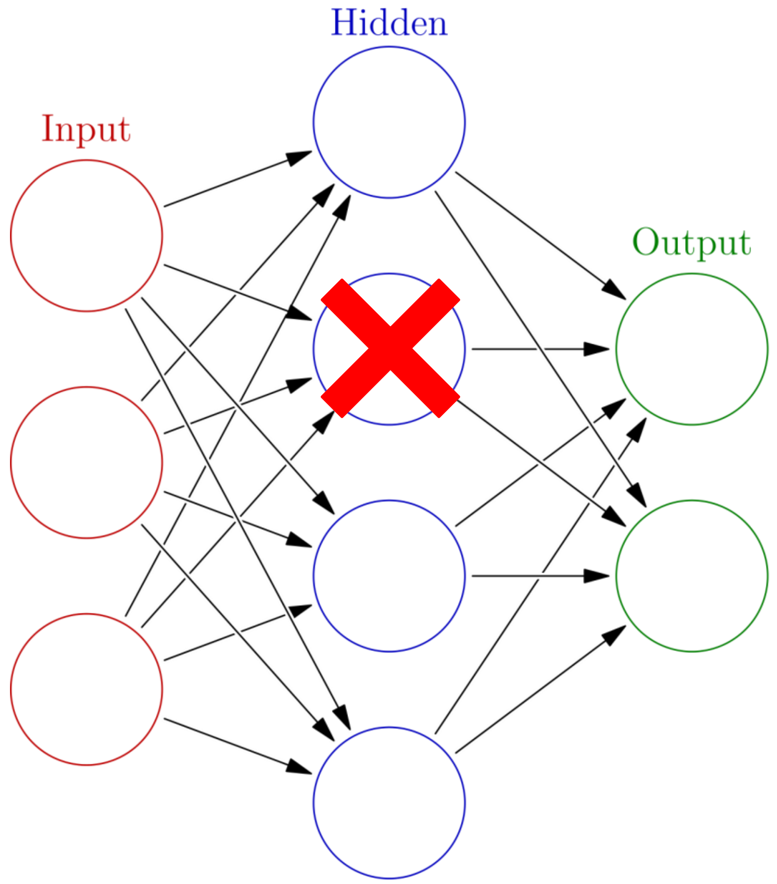
- Reducing the number of parameters (convolutional layers)
- Regularization

- There are other ways to prevent the model from overfitting to the training dataset, Dropout

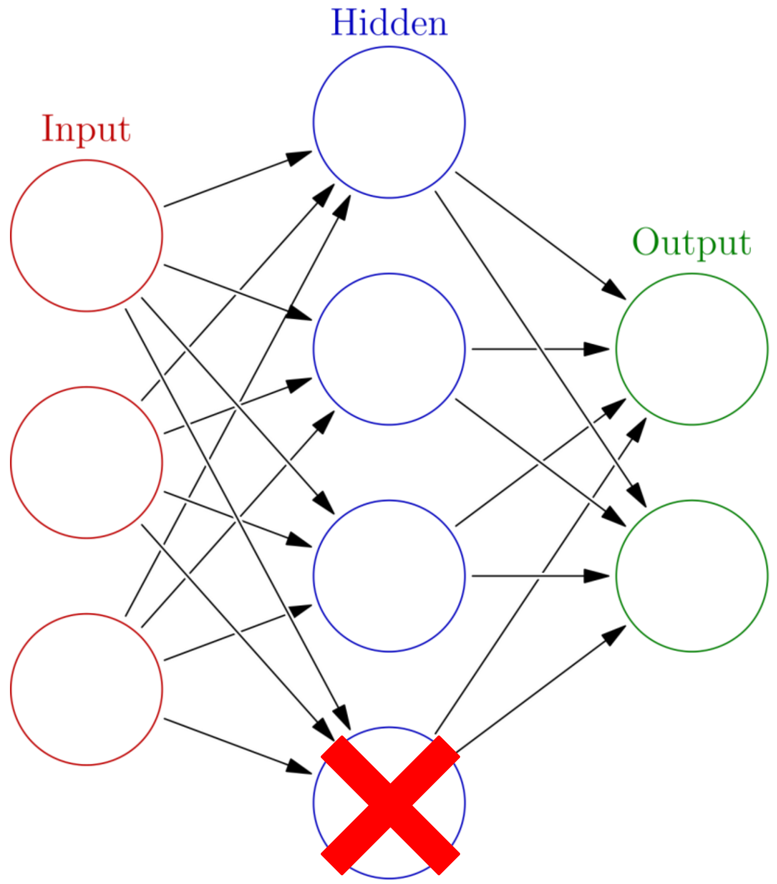
Dropout



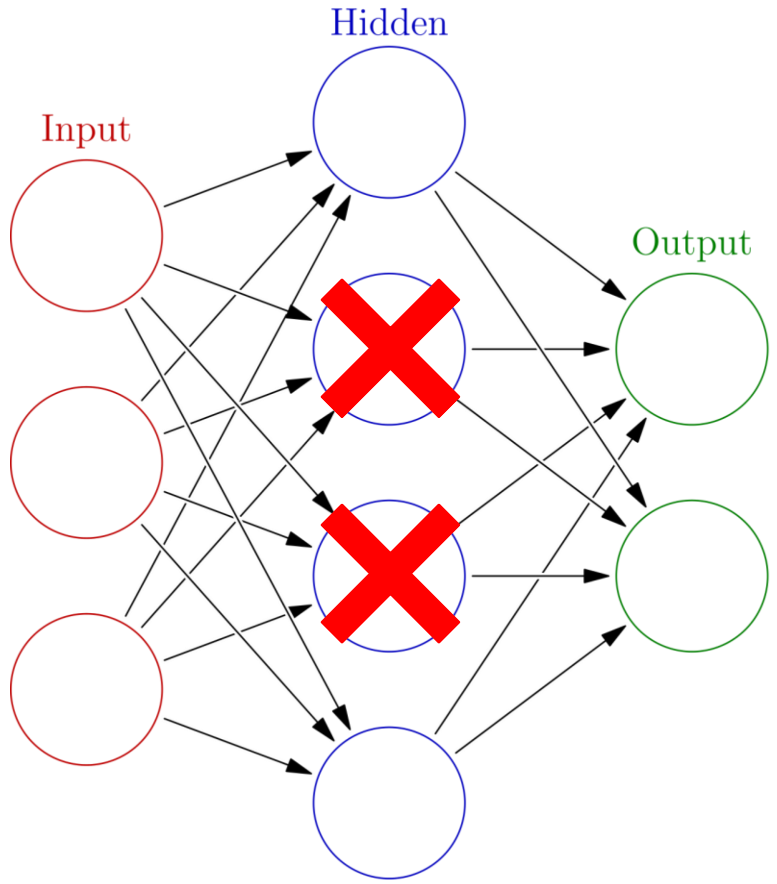
Dropout



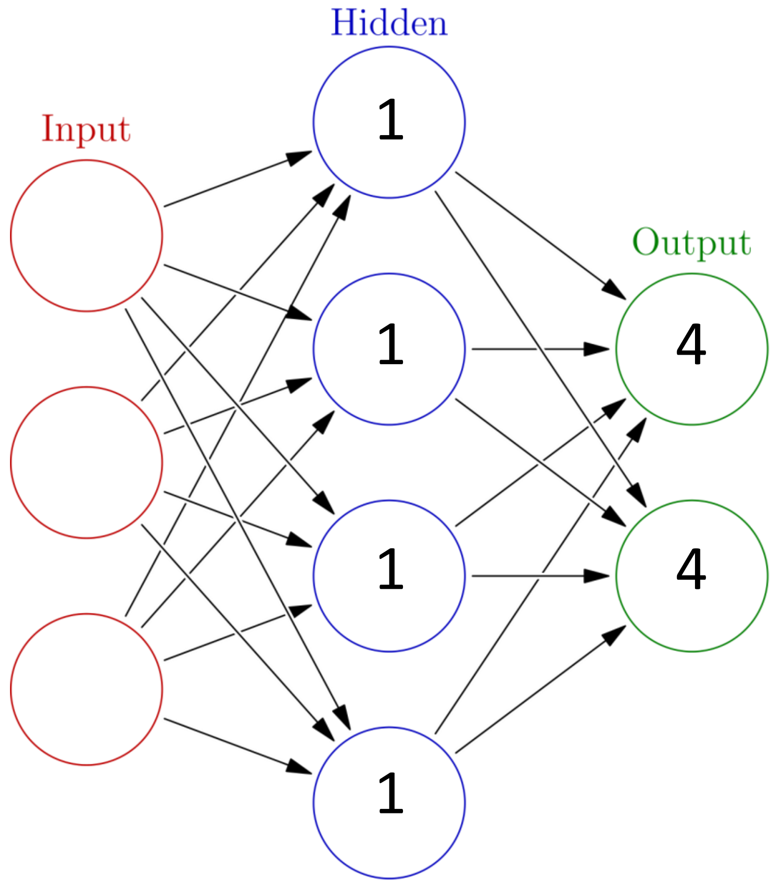
Dropout



Dropout

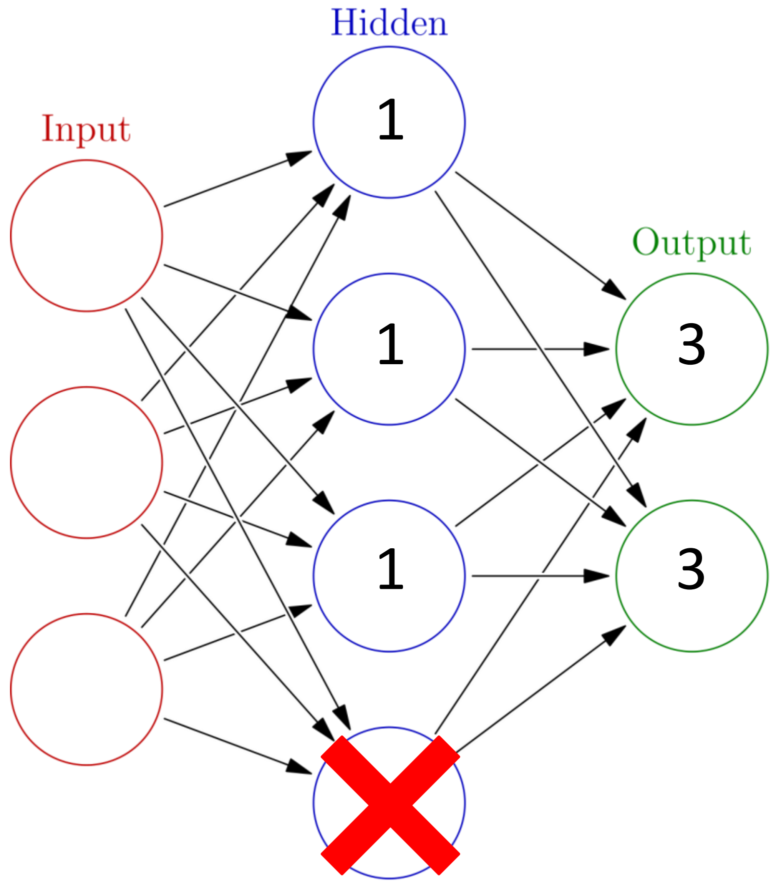


Dropout



Dropout

All weights are equal to one



Compensate changes in scale

Dropout

- Drop layer $d(x)$
- There are no parameters; the only hyperparameter is p (the probability of dropping a neuron). During the training phase:

$$d(x) = \frac{1}{p} m \circ x$$

Correct:(m is a vector of the same size as x; elements are sampled from the Bernoulli distribution Ber(p))

Division by p is done to preserve the overall scale of the outputs.

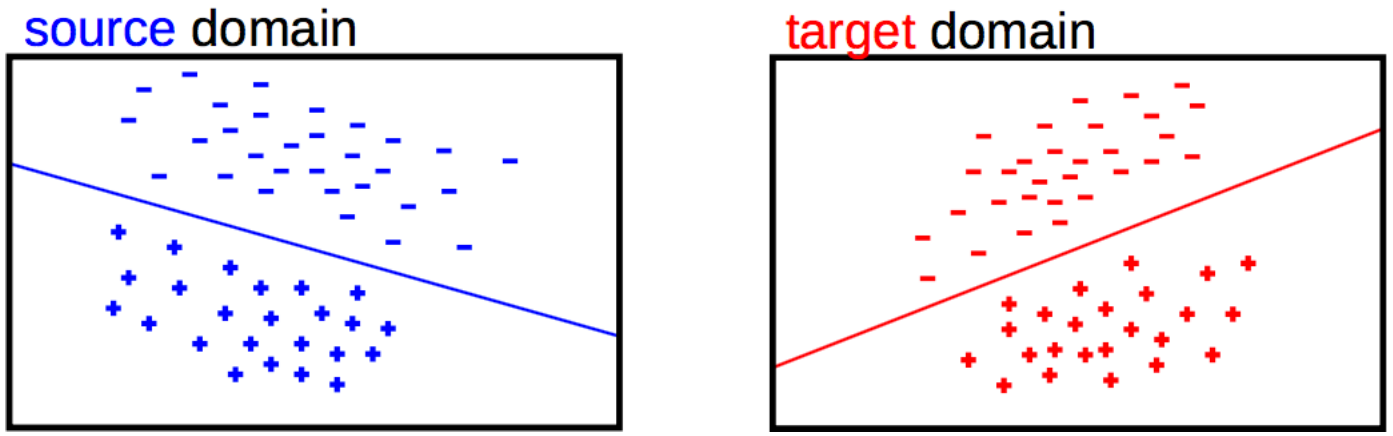
https://jmlr.org/papers/v15/srivastava14a.html

Dropout

- Interpretation: We train all possible neural network architectures derived from the original by dropping individual neurons
- All these architectures share the same weights
- During the application phase, we (almost) average the predictions of all these architectures

BatchNorm

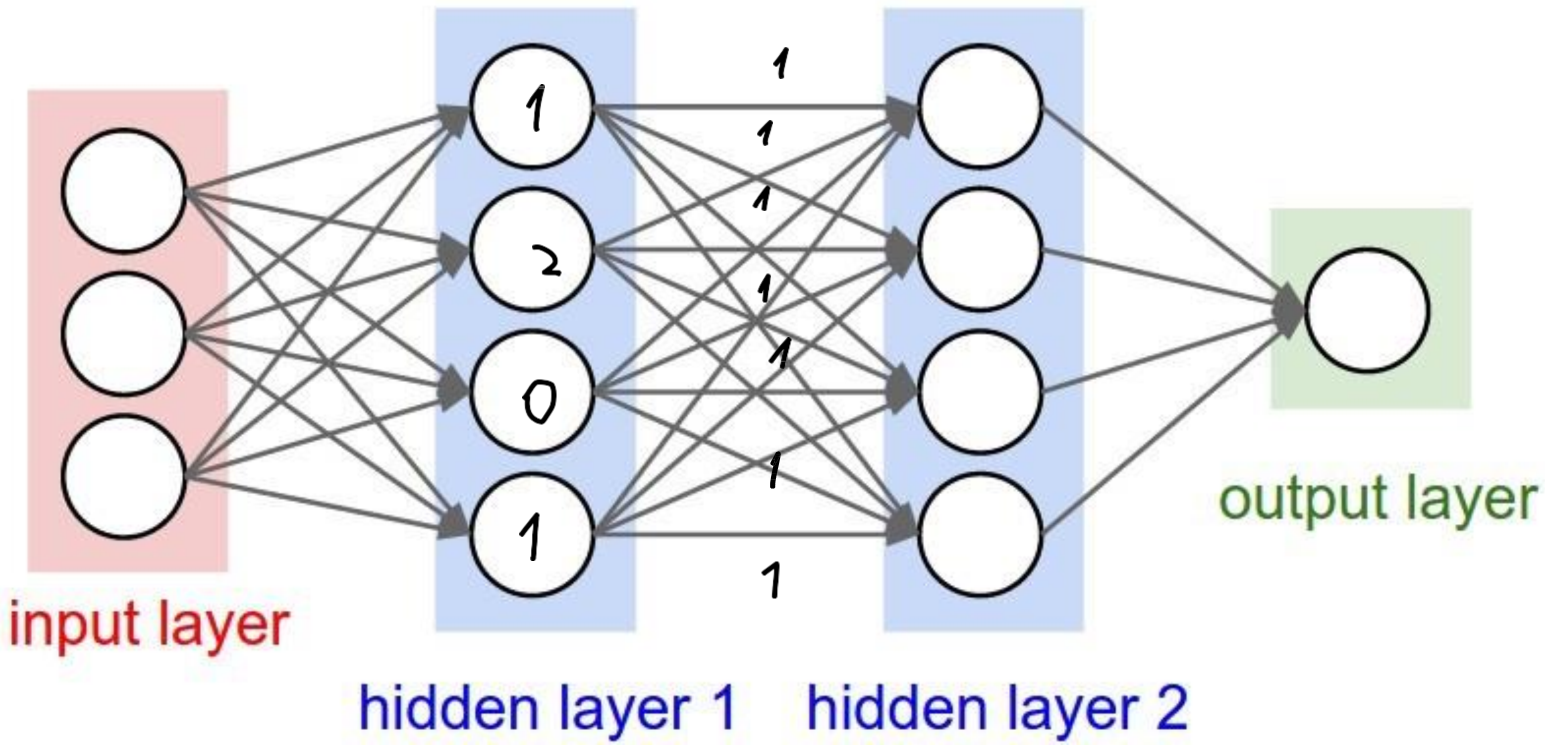
Covariate shift



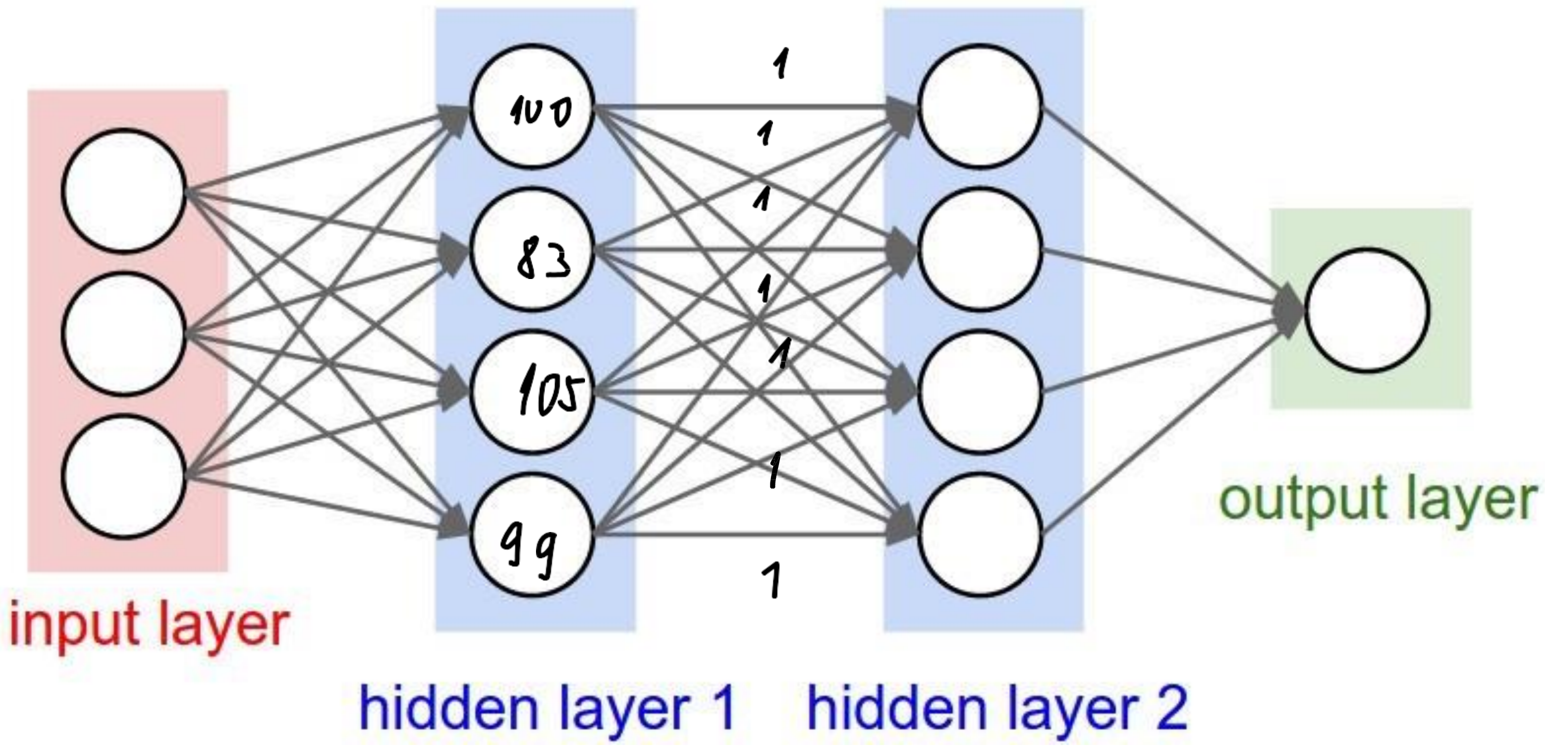
Internal covariate shift

- In a neural network, each layer’s input in output of the previous layer
- If a layer at the beginning undergoes significant changes, all subsequent layers need to be adjusted.

Internal covariate shift



Internal covariate shift



Internal covariate shift

- Idea: Transforming the outputs of layers to ensure they consistently have a fixed distribution
- BatchNorm

https://arxiv.org/pdf/1502.03167.pdf